2.6 — Why functions are useful, and how to use them effectively

ALEX[1] JANUARY 25, 2023

Now that we've covered what functions are and some of their basic capabilities, let's take a closer look at why they're useful.

Why use functions?

New programmers often ask, "Can't we just put all the code inside the *main* function?" For simple programs, you absolutely can. However, functions provide a number of benefits that make them extremely useful in programs of non-trivial length or complexity.

- Organization -- As programs grow in complexity, having all the code live inside the main() function becomes increasingly complicated. A function is almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to reduce a complicated program into smaller, more manageable chunks, which reduces the overall complexity of our program.
- Reusability -- Once a function is written, it can be called multiple times from within the program. This avoids duplicated code ("Don't Repeat Yourself") and minimizes the probability of copy/paste errors. Functions can also be shared with other programs, reducing the amount of code that has to be written from scratch (and retested) each time.
- Testing -- Because functions reduce code redundancy, there's less code to test in the first place. Also because functions are self-contained, once we've tested a function to ensure it works, we don't need to test it again unless we change it. This reduces the amount of code we have to test at one time, making it much easier to find bugs (or avoid them in the first place).
- Extensibility -- When we need to extend our program to handle a case it didn't handle before, functions allow us to make the change in one place and have that change take effect every time the function is called.
- Abstraction -- In order to use a function, you only need to know its name, inputs, outputs, and where it lives. You don't need to know how it works, or what other code it's dependent upon to use it. This lowers the amount of knowledge required to use other people's code (including everything in the standard library).

Effectively using functions

One of the biggest challenges new programmers encounter (besides learning the language) is understanding when and how to use functions effectively. Here are a few basic guidelines for writing functions:

- Groups of statements that appear more than once in a program should generally be made into a function. For example, if we're reading input from the user multiple times in the same way, that's a great candidate for a function. If we output something in the same way in multiple places, that's also a great candidate for a function.
- Code that has a well-defined set of inputs and outputs is a good candidate for a function, (particularly if it is complicated). For example, if we have a list of items that we want to sort, the code to do the sorting would make a great function, even if it's only done once. The input is the unsorted list, and the output is the sorted list. Another good prospective function would be code that simulates the roll of a 6-sided dice. Your current program might only use that in one place, but if you turn it into a function, it's ready to be reused if you later extend your program or in a future program.
- A function should generally perform one (and only one) task.
- When a function becomes too long, too complicated, or hard to understand, it can be split into multiple sub-functions. This is called **refactoring**. We talk more about refactoring in lesson 3.10 -- Finding issues before they become problems[2].

Typically, when learning C++, you will write a lot of programs that involve 3 subtasks:

1. Reading inputs from the user
2. Calculating a value from the inputs
3. Printing the calculated value

For trivial programs (e.g. less than 20 lines of code), some or all of these can be done in function *main*. However, for longer programs (or just for practice) each of these is a good candidate for an individual function.

New programmers often combine calculating a value and printing the calculated value into a single function. However, this violates the "one task" rule of thumb for functions. A function that calculates a value should return the value to the caller and let the caller decide what to do with the calculated value (such as call another function to print the value).

Next lesson
2.7 Forward declarations and definitions

3

Back to table of contents

4

Previous lesson
2.5 Introduction to local scope

5

6

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[9] are connected to your provided email address.

115 COMMENTS Newest

IceFloe
February 20, 2024 3:43 am

I think it's not worth making the code too modular and complicating where it's enough to add a couple of lines to the main function.

Everything is fine in moderation, and with parameters and copying values, it is still worth figuring out what depends on what and how to use it most effectively.

By the way, another interesting question is, is it possible to create two functions with the same name, but which will have different arguments and take different parameters, returning different values? example:

```
int Sum ( int x, int y)
{
return x+y;
}
int Sum (int z, int v, int c)
{
return z+v+c;
}
```

Last edited 2 months ago by IceFloe

0 Reply

Alex Author

Reply to IceFloe 10 February 22, 2024 1:36 pm

1. Mostly agree, but we're illustrating good practices here on short, simple problems. These practices will be more useful later as programs get more complex.
2. Yes, this is called function overloading. We have lessons on this later.

3 Reply

Pocket

February 15, 2024 6:17 am

```
#include <iostream>


int givAdd(int a, int b)
{
    return a + b;
}
int givSub(int a, int b)
{
    return a - b;
}
int givMul(int a, int b)
{
    return a * b;
}
int givDiv(int a, int b)
{
    return a / b;
}

int main()
{
    std::cout
        << "Two Numbers for math: ";
    int getA{};
    int getB{};
    int pickOpp{};

    std::cin
        >> getA
        >> getB;

    std::cout
        << "What kinda maths, 1 = add, 2 = sub, 3 = mul, 4 = div.\n"
        << "All other numbers will Div.\n";

    std::cin
        >> pickOpp;

    if (pickOpp <= 1)
    {
        std::cout
            << givAdd(getA, getB);
    }
    else if (pickOpp == 2)
    {
        std::cout
            << givSub(getA, getB);
    }
    else if (pickOpp == 3)
    {
        std::cout
            << givMul(getA, getB);
    }
    else
    {
        std::cout
            << givDiv(getA, getB);
    }

    return 0;
}
```

0 Reply

Pocket

Reply to Pocket 11 February 15, 2024 6:17 am

It's the perfect calculator.

0 Reply

alb

January 28, 2024 3:43 am

// I know this very obviously produces undefined behavior I am just curious as to why I can not use functions this way

```
#include <iostream>

int userIn()
{
std::cout << "Enter a number: " << '\n';

int x{};

std::cin >> x;

return x;

}

int userOut()
{
int x{};

int doubleNum();

std::cout << x << " doubled is : " << doubleNum() << '\n';

return doubleNum();

}

int doubleNum()
{
int x{};

return x * 2;

}

int main()
{
int userIn();
```

int doubleNum();

int userOut();

std::cout << userIn() << doubleNum() << userOut() << '\n';

return 0;

}

0 Reply

Patrick
Reply to alb 12 January 31, 2024 11:55 am

int doubleNum()
{
int x{};

return x * 2;
}

You double x without defining it to be some value, in some systems this will automatically format the variable to a default value. Essentially you can think of it as saying you want to double a number but don't tell me what number you want to double. How can I double a number that I don't know the value of?

0 Reply

BillMai
December 21, 2023 12:54 am

Thanks!!!

Last edited 4 months ago by BillMai

1 Reply

thisisfun
November 8, 2023 9:43 am

Didnt follow the DRY rule. Here i am still tho this is fun still i could have done namespace but i like this better

```
#include <iostream>



// Making a calculator...
int main()
{
    int x;
    int y;
    int z;
    int m;
    int n;
    int o;
    int p;
    int q;
    std::cout << "Today we will be making a Calculator... That goes to + - *
/ in a line...";
    std::cout << "+" << '\n';
    std::cin >> x;
    std::cin >> y;
    std::cout << "-" << '\n';
    std::cin >> z;
    std::cin >> m;
    std::cout << "*" << '\n';
    std::cin >> n;
    std::cin >> o;
    std::cout << "/" << '\n';
    std::cin >> p;
    std::cin >> q;
    std::cout << "Adding...: " << x + y << " Added!" << '\n';
    std::cout << "Now the -: " << z - m << '\n';
    std::cout << "Now the multiplying: " << n * o << '\n';
    std::cout << "Now the dividing: " << p / q << '\n';
    return 0;
}
```

This is Really fun making this

3 Reply

Jason E

Reply to thisisfun 13 December 30, 2023 12:26 am

```
int x, y, z, m, n, o, p, q;
```

This, I think would make it read better than initializing them all down a line, a lot faster to write as well.

2 Reply

Gripps

Reply to Jason E 14 January 11, 2024 6:10 pm

Good suggestion! By the way, how do you make your text appear as though it's in an

IDE?

Last edited 3 months ago by Gripps

0 Reply

aaa
Reply to Gripps [15] January 19, 2024 10:27 am

The built in terminal

1 Reply

Dedi
October 23, 2023 11:03 pm

can we make a function as another file? and how to connect to that function file?

0 Reply

Alex Author
Reply to Dedi [16] October 24, 2023 9:59 am

Keep reading. Covered in lesson https://www.learncpp.com/cpp-tutorial/programs-with-multiple-code-files/

2 Reply

Serif
October 1, 2023 12:55 am

Why do we need to know the input and outputs of a function to use it, as its said on abstraction part, i probably didnt understand that well, what do we mean by knowing its outputs and inputs?

0 Reply

Alex Author
Reply to Serif [17] October 2, 2023 1:11 pm

Inputs are the values you give to the function to work with.
Outputs are the values that the function gives back to you.

To use a function, you need to know what kind of inputs it takes, and what those inputs actually mean. Same with the outputs.

To use an analogy, think of a vending machine. In order to use the machine, you need to know what kind of money it takes and what the denominations of the money is. You also have to know how the vending machine is going to return the item you're purchasing back to you as well. If you don't know any of these things you're going to have trouble using the vending machine in any sort of meaningful way.

2 Reply

vstar
August 6, 2023 8:46 pm

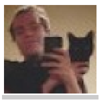
哈哈太棒了，就好比什□是□□，什□是□□，反正我□得那就是天□，□□作者□我解□了天□

4 Reply

Kvicala
May 1, 2023 2:36 am

It's nice that they're explaining why functions are useful. I wish someone did that when I was learning Java, to this day I have no goddamn clue as to what is interfaces', abstract methods' and abstract classes' purpose. I feel that there are just some legacy control tool, which you can handle by documentation.

1 Reply



Taste
April 5, 2023 10:21 am

This is really cool, here I split up all the work to be in functions, leaving only 4 lines in my main function.

Loving this course

```cpp
#include <iostream>

int getNum()
{
    int num{ };
    std::cout << "Enter a number: ";
    std::cin >> num;

    return num;
}

int doubleNumber(int x)
{
    return x * 2;

}

void printMath(int x, int y)
{
    std::cout << x << " doubled is " << doubleNumber(x) << '\n'
              << y << " doubled is " << doubleNumber(y) << '\n';
}

int main()
{
    int x{ getNum() };
    int y{ getNum() };
    printMath(x, y);

    return 0;
}
```

Last edited 1 year ago by Taste

8 Reply

Docksie

Reply to Taste 18 April 26, 2023 10:43 pm

could reduce it by 2 more lines:

printMath(getNum(), getNum();

4 Reply

dyderf

Reply to Docksie 19 June 21, 2023 12:38 pm

Although in this example it doesn't matter that much, this is actually undefined behavior. The first number typed by the user could be printed as the second and vice versa.

1 Reply

resident of flavourtown

Reply to dyderf 20 September 3, 2023 9:24 am

Could you explain why that is? To me, it seems that the first input would always be x, and so always be printed first;
though I may be missing something.

0 Reply

Alex Author

Reply to resident of flavourtown 21 September 5, 2023 12:25 pm

Discussed in lesson https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/

1 Reply

Wanderer above the Sea of Fog

Reply to resident of flavourtown 21 September 5, 2023 5:40 am

by using this
"printMath(getNum(), getNum();"

this instead of this

"int x{ getNum() };
int y{ getNum() };
printMath(x, y);"

in the main function x and y are not created and initialized.

1 Reply

Krishnakumar

January 23, 2023 1:54 am

Ridiculously minor nitpick:

>making it much easier to find bugs (or avoid them in the first place).

Experience suggests that for any non-trivial function, we can never fully **avoid** bugs (such is life), but can attempt to minimize them as much as reasonably possible by following such good practices.

0 Reply

Emeka Daniel
Reply to Krishnakumar 22 March 17, 2023 5:50 am

You must have read that wrong or maybe it's just based on how to understand the word avoid.

0 Reply

Krishnakumar
Reply to Emeka Daniel 23 August 24, 2023 1:15 am

Indeed. Re-reading it helped!

0 Reply

Emeka Daniel
Reply to Krishnakumar 24 August 24, 2023 3:00 am

Okay.

0 Reply

Alex Author
Reply to Krishnakumar 22 January 25, 2023 11:02 am

Hmmm, I read that as "making it easier to... avoid them in the first place", which doesn't imply they can all be avoided.

6 Reply

Krishnakumar
Reply to Alex 25 May 25, 2023 4:39 pm

Thanks. That helps

0 Reply

silent
December 14, 2022 5:13 am

"Extensibility -- When we need to extend our program to handle a case it didn’t handle before, functions allow us to make the **change in one place** and have that change take effect every time the function is called."
"Make the change" where exactly? In the function itself or in the new piece of code that is affected by this function, or both? What kind of change? Same for the "take effect" part.

0 Reply

Alex Author
Reply to silent [26] December 17, 2022 4:25 pm

Make the change in the body of the function. Whenever the function is executed, the change will be executed transparently to the caller.

For example, consider this function

```
void foo()
{
}
```

When called, it does nothing. Now let's add a change:

```
void foo()
{
    std::cout << "Moo";
}
```

Now whenever it is called, it will print "Moo". The code on the caller's side doesn't need to change at all.

6 Reply

Tiziano
November 11, 2022 1:45 pm

Great tutorial Alex! Thanks!
I have a question: is the standard library available to be opened to look inside its code? if yes, how?

2 Reply

Alex Author
Reply to Tiziano [27] November 13, 2022 5:44 pm

Only partially. You can open the header files and see what's inside those (they will probably not be very intelligible). The code defined inside .cpp files is precompiled into a format that's non-viewable.

1 Reply

Krishnakumar
Reply to Alex [28] May 25, 2023 4:43 pm

Here's the source code for the standard library of one conforming C++ compiler implementation: https://github.com/llvm/llvm-project/tree/main/libcxx/ . You can take a look at the implementation details of all the standard library functions.

Last edited 10 months ago by Krishnakumar

1 Reply

South
April 26, 2022 1:28 am

Not sure if the way I think isn't normal but functions I think are the best way to go. A dice roll function can be used by a game to calculate random damage to the player or mob but can also be used in any other program where a random number is desired.

I see functions as modules to be plugged in as needed and accessed. I see coding in general as a group of modules working together through linking code in the main to produce a result.

8 Reply

Alex Author
Reply to South 29 April 26, 2022 9:37 am

Exactly. If you're smart about how you design your "modules" they will be reusable across many different programs. This is maximized by separating the logic of your program (which is specific to your program) from the reusuable components (functions and types that can be used in any program).

13 Reply

KKW
December 24, 2021 6:45 am

so like the brawl star figures, they were made using functions and if else so they will attack when the attack button is pressed, right? and the COC troops keep attacking and don't attack when walking is also because of the use of functions and if else, right?

5 Reply

Ankesh
Reply to KKW 30 August 29, 2022 3:52 am

yeah you are right

1 Reply

MWAR
Reply to KKW [30] March 3, 2022 7:54 pm

Not sure what you're referring to. But keep it up! Learning by relating to things you already know is one of the best ways to retain information! I use the ADEPT method : https://betterexplained.com/articles/adept-method/

4 Reply

Jean Peter T. Paredes
August 26, 2021 6:23 pm

"Statements that appear more than once in a program should generally be made into a function. For example, if we're reading input from the user multiple times in the same way, that's a great candidate for a function. If we output something in the same way multiple times, that's also a great candidate for a function."

Would like to add that

Statements that appeared once BUT has a high probability of being used multiple times in the future should generally be made into a function
just like the get user input from our previous lessons

10 Reply

Nitin
August 22, 2021 9:27 pm

isn't it okay to use "using namespace std;" than using every time std: with in-build functions.

1 Reply

Alex Author
Reply to Nitin [31] August 25, 2021 12:44 pm

This question is addressed in lesson https://www.learncpp.com/cpp-tutorial/using-declarations-and-using-directives/

10 Reply

Ni3
Reply to Alex [32] August 26, 2021 1:38 am

Okay thank you sir

4 Reply

lil pump
February 20, 2021 7:36 am

[C++]
#include iostream

int main(){
  std::cout << "Hello world" << std::endl;
}

[/C++]

0 Reply

seaque
Reply to lil pump 33 June 1, 2021 6:40 am

the correct use is

```
#include iostream

int main(){
    std::cout << "Hello world" << std::endl;
}
```

0 Reply

Vinorosso
Reply to seaque 34 June 21, 2021 2:03 am

The code should be like this tho:

Copy

```
#include <iostream>

int main()
{
    std::cout << "Hello world\n";

    return 0;
}
```

0 Reply

seaque
Reply to seaque 34 June 1, 2021 6:58 am

sorry, i meant (code) (/code)

change the parenthesis with []

1 Reply

RHOULAN DHAMAR WANTO
November 21, 2020 11:12 pm

please help Alex
please help me with this

(Equilateral Triangle validation and perimeter) Implement the following two functions:
// Returns true if all the sides of the triangle
// are same.
bool isValid(double side1, double side2, double side3)
// Returns the perimeter of an equilateral triangle.
double perimeter(double side1)
The formula for computing the perimeter is perimeter = 3 * side. Write a test program that reads three sides for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid.

0 Reply

Anon
Reply to RHOULAN DHAMAR WANTO 35 December 17, 2020 10:56 am

```
#include <iostream>

bool isValid(double side1, double side2, double side3)
{
    //if side1 == side2 then they have the same length, and if side2 and side 3 have the
same length,
    //it follows that side1 == side2 == side3 and therefore it is equilateral
    return (side1 == side2) && (side2 == side3);
}

double perimeter(double side1)
{
    return side1 * 3;
}

int main()
{
    //you could refactor a way to take the input for all 3 sides as a function
    std::cout << "Please enter side1 of your triangle:";
    double side1{};
    std::cin >> side1;
    std::cout << "Please enter side2 of your triangle:";
    double side2{};
    std::cin >> side2;
    std::cout << "Please enter side3 of your triangle:";
    double side3{};
    std::cin >> side3;
    if (isValid(side1, side2, side3))
    {
        std::cout << "The perimeter of an equilateral triangle with side length " << side1 << " is
" << perimeter(side1) << ".";
    } else
    {
        std::cout << "The triangle is not equilateral.";
    }
    return 0;
}
```

2 Reply

John
November 10, 2020 6:13 am

for practice purposes:

```cpp
#include <iostream>

int get_value()
{
    int v{};

    std::cout << "Enter a Number: ";
    std::cin >> v;

    return v;
}

char get_operator()
{
    char o{};

    std::cout << "Enter operator: ";
    std::cin >> o;

    return o;
}

int main()
{
    int a{ get_value() };
    char op{ get_operator() };
    int b{ get_value() };

    switch (op)
    {
        case '*':
            std::cout << a * b << "\n";
            break;
        case '/':
            std::cout << a / b << "\n";
            break;
        case '+':
            std::cout << a + b << "\n";
            break;
        case '-':
            std::cout << a - b << "\n";
            break;
    }

    return 0;
}
```

4 Reply

lettuceMan

Reply to John 36 February 14, 2022 7:37 am

nice

2 Reply

Mathari
May 9, 2020 9:00 pm

Hello
Are there any tutorials or exercises accompanying each lesson?

0 Reply

José
Reply to Mathari May 19, 2020 6:46 am

If you search on the internet, maybe you'll find something

0 Reply

nascardriver Sub-admin
Reply to Mathari May 10, 2020 3:23 am

Not for each lesson. There's a quiz every now and then.

0 Reply

you're using a function provided by the standard library
May 5, 2020 8:43 am

```
"Although it doesn't look like it, every time you use operator<< or
operator>> to do input or output, you're using a function provided by the
standard library that meets all of the above criteria."
```

I found this part very interesting. Can we reach in this tutorial to a point where we can start writing user-defined functions with such characteristic you mentioned in the quote?
I mean I hadn't had any idea std::cout, could be a function!

0 Reply

nascardriver Sub-admin
Reply to you're using a function provided by the standard library May 6, 2020 6:40 am

`std::cout` is a variable, the function is `<<`. We show how to write those functions in the chapter about operator overloading.

1 Reply

ColdCoffee
April 7, 2020 11:02 am

"A function should generally perform one (and only one) task."
If function always performs one and only one task then why do you use "generally" ?

0 Reply

> **Gabe**
> Reply to ColdCoffee April 9, 2020 1:32 pm
>
> The idea is to use a function that only solves a single computational. No more. If you have a function that not only computes the sum of two parameters passed as arguments, but also prints out the result in the same function, that only shows that you can separate the two as their own functions. Functions should be short and sweet and binds to the name of the function.
>
> 1 Reply

Jonas
June 9, 2019 12:58 am

Found a typo at the first point of 'Effectively using functions' (incorrect word is marked with asterisks and already corrected):

Statements that *appear* more than once in a program should generally be made into a function.

0 Reply

Rowan
February 28, 2019 12:16 pm

do you know why 0 does not move
#include <iostream>
#include <conio.h>

using namespace std;
bool gameover;
const int width = 20;
const int height = 20;
int x, y, fruitX,fruitY, score;
enum eDirection { STOP = 0, LEFT, RIGHT, UP, DOWN};
eDirection dir;
void Setup() {
  gameover = false;

```
    dir = STOP;
    x = width / 2;
    y = height / 2;
    fruitX = rand() % width;
    fruitY = rand() % height;
    score = 0;
}
void Draw() {

    system("cls");
    for (int i = 0; i > width; i++)
        cout << "#";
    cout << endl;

    for (int i = 0; i < height; i++)
    {

        for (int j = 0; j < width; j++)
        {

            if (j == 0)

                cout << "#";
            if (i == y && j == x)
                cout << "0";

            else if (i == fruitY && j == fruitX)
                cout << "F";

            else




                cout << " ";
            if (j == width + 2)
                cout << "#";
        }
        cout << endl;
    }

    for (int i = 0; i > width+2; i++)
        cout << "#";
    cout << endl;





}
void input() {
    if (_kbhit())
    {
        switch (_getch()) {
        case 'a':
            dir = LEFT;
            break;
        case 'd':
            dir = RIGHT;
            break;
        case 'w':
            dir = UP;
            break;
        case 's':
            dir = DOWN;
            break;
        case 'x':
            gameover = true;
            break;
        }
    }
}
void Logic()
{
    switch (eDirection())
    {
    case STOP:
        break;
    case LEFT:
        break;
    case RIGHT:
        break;
    case UP:
        break;
    case DOWN:
        break;
    default:
        break;
    }
    if (x > width || x < 0 || y > height || y < 0)
        gameover = true;
}

int main()
{
```

Setup();
while (!gameover)
Draw();
input();
Logic();

return 0;
}

0 Reply

Julius
Reply to Rowan March 31, 2019 7:28 am

I did not read in detail through all of it but one problem I found was that the GameLoop in Main (while(!gameover))
only calls your draw function, not the input and Logic functions. They only get called once the gameLoop is over.
I would suggest moving the input and Logic funtion calls into the GameLoop and see what happens
(change this)

```
while(!gameover) {
    Draw();
}
input();
Logic();
```

to this

```
while(!gameover) {
    input();
    Logic();
    Draw();
}
```

0 Reply

Anonymous
November 8, 2018 5:11 am

I combined all the things I learnt so far up to this point
Is there anything I could improve on in this code or so far so good?

#include <iostream>

void doNothing(const int &q)
{

```
}

int getValueFromUser()
{
    std::cout << "Enter a number: " << std::endl;
    int a;
    std::cin >> a;
    return a;
}

void doRendering()
{
    std::cout << "Rendering..." << std::endl;
}

void doConfiguring()
{
    std::cout << "Configuring..." << std::endl;
}

void doRenderAndConfigure()
{
    doRendering();
    doConfiguring();
}

void printValue(int x, int y)
{
    std::cout << x << std::endl;
    std::cout << y << std::endl;
}

int add(int x, int y)
{
    return x + y;
}

int multiply(int x, int y)
{
    return x * y;
}

int doubleNumber(int x)
{
    return x * 2;
}
```

```
int Return5()
{
   return 5;
}

int main()
{
   int x = getValueFromUser();
   int y = getValueFromUser();
   int z;
   doNothing(z);
   doRenderAndConfigure();
   std::cout << "Hello World!\nYour numbers are: " << add(doubleNumber(x), y) << std::endl;
   std::cout << z << std::endl;
   printValue(Return5(), multiply(Return5(), 2));
   return 0;

}
```

0 Reply

nascardriver
Reply to Anonymous November 8, 2018 7:46 am

- "q" is a poor variable name. Avoid abbreviations unless they're obvious.

Other than that there's not much to say, because this is lesson 1.4b. You'll learn about other improvements in future lessons.
When you're done with lesson 2.7, there should two improvements you can make to your code. If you don't find them, feel free to repost your code or leave a reply here.

0 Reply

Dhruv Tyagi
April 19, 2018 12:39 pm

Dear Alex, I made a small calculator to add, subtract, multiply and devidetwo integers...Can you edit the code such that it can calculate the decimals too?

```
#include "stdafx.h"
#include <iostream>

int add(int x, int y) // add two numbers
{
    return x + y;
}

int multiply(int x, int y) // multiply two numbers
{
    return x * y;
}

int subtract(int x, int y) // subtract two numbers
{
    return x - y;
}

int devide(int x, int y) // devide two numbers
{
    return x / y;
}

int req1()  //choose the first number
{
    int a;
    std::cout << "Enter the first variable: " << std::endl;
    std::cin >> a;
    return a;
}

int req2()   //choose the second number
{
    int b;
    std::cout << "Enter the second variable: " << std::endl;
    std::cin >> b;
    return b;
}

int req3() //choose the operator
{
    int op;
    std::cout << "Enter the mathematical operator: (1 = + , 2 = - , 3 = * ,
4 = /) " << std::endl;
    std::cin >> op;
    return op;
}

void calculate() {   //calculate
    int p, q, r;
    p = req1();  // number 1
    q = req2();  //number 2
    r = req3();  // operator
    if (r == 1) {  // add
        std::cout << p << " + " << q << " = " << add(q, p) << std::endl;
    }
    else if (r == 2) {   // subtract
        std::cout << p << " - " << q << " = " << subtract(p, q) <<
std::endl;
    }
    else if (r == 3) {  //multiply
        std::cout << p << " * " << q << " = " << multiply(p, q) <<
std::endl;
    }
    else if (r == 4) {  // devide
        std::cout << p << " / " << q << " = " << devide(p, q) << std::endl;
        std::cout << p << " / " << q << " = " << devide(p, q) << std::endl;
    }
    else {  // invalid operator
        std::cout << "Enter a valid operator..." << std::endl;
    }
}
int main() {
    calculate();
}
```

Thanks

0 Reply

codePhobia

Reply to Dhruv Tyagi December 25, 2022 3:21 pm

```
#include <iostream>

double getNum()
{
    double inp{};
    std::cin >> inp;
    return inp;
}

int main()
{
    std::cout << "Enter the first variable: ";
    double num1{ getNum() };
    std::cout << "Enter the second variable: ";
    double num2{ getNum() };
    std::cout << "Enter the mathematical operator: (1 = +, 2 = -, 3
= *, 4 = /\n";
    int op{};
    std::cin >> op;

    switch (op)
    {
        case 1:
            std::cout << num1 << " + " << num2 << " = " << num1 +
num2;
            break;
        case 2:
            std::cout << num1 << " - " << num2 << " = " << num1 -
num2;
            break;
        case 3:
            std::cout << num1 << " * " << num2 << " = " << num1 *
num2;
            break;
        case 4:
            std::cout << num1 << " / " << num2 << " = " << num1 /
num2;
            break;
        default:
            std::cout << "Enter a valid operator...\n";
    }

    return 0;

    /*
    if (op == 1)
    {
        std::cout << num1 << " + " << num2 << " = " << num1 + num2;
    }
    else if (op == 2)
    {
        std::cout << num1 << " - " << num2 << " = " << num1 - num2;
    }
    else if (op == 3)
    {
        std::cout << num1 << " * " << num2 << " = " << num1 * num2;
    }
    else if (op == 4)
    {
        std::cout << num1 << " / " << num2 << " = " << num1 / num2;
    }
    else
    {
        std::cout << "Enter a valid operator...\n";
    }
    */
}
```

0 Reply

nascardriver

Reply to Dhruv Tyagi April 20, 2018 8:36 am

Hi Dhruv!

Use 'double' instead of 'int'.

References
Lesson 2.1 - Fundamental variable definition, initialization, and assignment
Lesson 2.5 - Floating point numbers

0 Reply

Singh

Reply to nascardriver June 18, 2019 10:08 am

Hi there, How's it going?

Can you please look into my case. Please check with line number 11 and 56.

1. Unable to use double to get decimal value on a division of number.
2. Not able to clear screen if not a valid input from a user.

```cpp
//Program is to use different functions to get the results.

#include <iostream>

int addfunction(int a, int b)
    { return a+b; }

int multi(int a, int b)
    { return a*b; }

int division(int a, int b) //not getting result even using double.
    { double c=a/b;
        std::cout<<"Division of both A & B is: " <<c;
        return c; }

int main()
    {
        std::cout<<"Program is to add, mulitply or divide the numbers.\n";
        int x{};
        std::cout<<"Please select the function.\nTo ADD: 1\nTo MUL: 2\nTo DIV: 3\n\nInput: ";
        std::cin>>x;

        // And how can I use the below set of code in new function which can be called in if Statements?
        //only after condition check whereas it has to run every time. But its working fine If I put
        //"else if(x>=4)" condition as "if(x>=4)" below (before a & b declaration.) Such as Below:
        //    if(x>=4)
        //      {
        //          std::cout<<"Please input valid number.\n";
        //          std::cin.clear();
        //          main();
        //      }
        //    Is it a good approach or do we have any solution to it. */

        int a, b;
        std::cout<<"\nPlease enter the numbers.";
        std::cout<<"A: ";
        std::cin>>a;
        std::cout<<"B: ";
        std::cin>>b;

        if(x==1)
        {
            std::cout<<addfunction(a,b);
        }
            else if(x==2)
            {
                std::cout<<multi(a,b);
            }
             else if(x==3)
             {
                 std::cout<<division(a,b);
             }
                else if(x>=4)
                {
                    std::cout<<"Please input valid number.\n";
                    system("cls"); //how to perform clear scream and again it goes back to main after screen went
black, not working with clrscr
```

```
                    main();
                }

        return 0;
    }
```

0 Reply

nascardriver
Reply to Singh June 19, 2019 3:24 am

- Use your editor's auto-formatting feature.
- `main` cannot be used. You need a loop, loops are covered later.
- Don't use `system`, it won't work on other platforms.

> Line 12
`c`'s type doesn't matter. You're dividing an `int` by an `int`, the result is an `int`, which is then converted to a `double`. You need to copy `a` or `b` or both into a double and divide that. (You'll learn about casts later, you don't need an extra variable then).

> Line 56
There is no universal way to clear the console/terminal. You can try hiding the previous output by printing a lot of line feeds.

```
std::cout << std::string(1024, '\n'); // Repeats '\n' 1024 times.
// I made up 1024. There is no universal way of retrieving the terminal height.
```

1 Reply

Henry
April 4, 2018 7:38 am

Hi Alex,please what does these three statements do:

"std::cin.clear();
std::cin.ignore(32767, '\n');
std::cin.get();" ????

0 Reply

Anonymous
Reply to Henry April 20, 2018 8:04 am

you can use double instead of int to define all the variables and you will be able to do decimal calculations as well.

0 Reply

nascardriver
Reply to Henry April 4, 2018 7:46 am

Hi Henry!

This is covered in lesson 5.10.

@std::cin.clear clears @std::cin's error flag
@std::cin.ignore ignores all characters in the input stream until '\n' is found, but a maximum of 32767 characters.
@std::cin.get reads one character from the input stream

0 Reply

Henry
Reply to nascardriver April 4, 2018 11:46 pm

Thank you for this tutorial.
I used to be scared of programming before now, but this tutorial has changed that.

0 Reply

ScarLet
July 6, 2017 8:49 pm

Is in a program only contain one function main() or can be more in a program?

0 Reply

Alex Author
Reply to ScarLet July 6, 2017 11:19 pm

A program can only contain one main() function, but may contain as many other functions as you want.

0 Reply

Kamran
July 6, 2017 3:38 am

I want to learn c++

1 Reply

UncleMi

Reply to Kamran July 7, 2017 8:35 am

You are in the right place then

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/finding-issues-before-they-become-problems/
3. https://www.learncpp.com/cpp-tutorial/forward-declarations/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/introduction-to-local-scope/
6. https://www.learncpp.com/why-functions-are-useful-and-how-to-use-them-effectively/
7. https://www.learncpp.com/cpp-tutorial/range-based-for-loops-for-each/
8. https://www.learncpp.com/cpp-tutorial/null-pointers/
9. https://gravatar.com/
10. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-593854
11. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-593650
12. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-592978
13. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-589534
14. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-591514
15. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-592206
16. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-589059
17. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-588012
18. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-579034
19. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-579838
20. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-582239

21. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-586517
22. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-576575
23. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-578473
24. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-586043
25. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-576703
26. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-575265
27. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-574626
28. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-574664
29. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-568076
30. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-563811
31. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-560489
32. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-560593
33. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-500748
34. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-536527
35. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-482250
36. https://www.learncpp.com/cpp-tutorial/why-functions-are-useful-and-how-to-use-them-effectively/#comment-480000